NASA-TM-111480

7N-63-TM
48082

Planning for Control

MARK DRUMMOND
JOHN BRESINA

AI RESEARCH BRANCH, MAIL STOP 244-17
NASA AMES RESEARCH CENTER
MOFFETT FIELD, CA 94035

NASA Ames Research Center
Artificial Intelligence Research Branch

Technical Report FIA–91–18

July 1990

Planning for Control

Mark Drummond[*] **and John Bresina**[†]
Sterling Federal Systems
AI Research Branch, NASA Ames Research Center
Mail Stop: 244-17, Moffett Field, CA 94035 U.S.A.

July 1990

Abstract

The study of automated planning within traditional artificial intelligence has produced a useful set of techniques for synthesizing plans of action. This traditional study of planning has concentrated on the problem of plan synthesis and, as a result, has left unaddressed many important issues involved with closed-loop plan execution. In contrast, traditional control theory has produced a set of analytical tools for the construction of robust control systems. This paper addresses problems that lie at the boundary of planning and control, from an AI planning perspective. Such problems involve the active selection of relevant actions through automated planning and also require the robust execution of the actions thus selected.

This paper appears in
The Proceedings the Fifth IEEE International Symposium on Intelligent Control,
published by the IEEE Computer Society Press.
(Philadelphia, PA; September 5–7, 1990)

[*]This work has been partially supported by the Artificial Intelligence Research Program of the Air Force Office of Scientific Research.

[†]Also affiliated with the Computer Science Department at Rutgers University.

Planning for Control

Mark Drummond[‡] and **John Bresina**[§]
Sterling Federal Systems
AI Research Branch, NASA Ames Research Center
Mail Stop: 244-17, Moffett Field, CA 94035

Abstract

The study of automated planning within traditional artificial intelligence has produced a useful set of techniques for synthesizing plans of action. This traditional study of planning has concentrated on the problem of plan synthesis and, as a result, has left unaddressed many important issues involved with closed-loop plan execution. In contrast, traditional control theory has produced a set of analytical tools for the construction of robust control systems. This paper addresses problems that lie at the boundary of planning and control, from an AI planning perspective. Such problems involve the active selection of relevant actions through automated planning and also require the robust execution of the actions thus selected.

Introduction

According to the field of Artificial Intelligence, *planning* is the task of selecting and sequencing actions. Actions are selected on the basis of their goal-achieving abilities, and are subsequently sequenced to ensure that the goals for which the actions have been selected are indeed achieved. The output from an AI planner is a plan, generally represented as a partially ordered set of actions. The processes of plan generation and plan use are typically quite distinct in the AI view; once a complete plan has been generated, it is handed off to a separate system for execution. The execution system is expected to follow the plan of action if possible. If the plan works, the original goal, as posed to the planner, is achieved; if the plan fails, the plan executor reports this back to the planner and the cycle of plan generation and execution is repeated.

Together, the planner and executor can be viewed as a "control system" that accepts a goal and repeatedly attempts to reach a state in which this goal is satisfied. For a variety of reasons, such a system will not work for a usefully general range of problems. This paper explains why. The next section outlines the capabilities of current plan generation and execution systems. Following this, a sample planning and control problem is defined and then used to show where traditional AI planning fails. The failure of traditional planning is used to motivate a discussion on aspects of a new approach to the relationship between plan generation and plan execution.

Traditional AI Planning and Execution

A traditional AI planner is given a variety of information from which it is expected to manufacture plans. It is given a set of operator schemata describing actions that can be taken by the execution system, a set of facts describing the execution system's current environment, and a goal to achieve. Operators are typically expressed as partial functions from one set of facts, or *initial state*, to another set of facts, or *successor state*. While the state-based method is not the only way of expressing operators, it has received significant study and use in the literature.

A *goal* is a predicate that must be made true at some point in the future. Typically, the predicate is a conjunction of non-negated facts, and the goal is said to be *achieved* by a plan when the execution of the plan can produce a state in which the predicate will be true. The plan generated by a planner must achieve the given goal, and additionally, it must be composed of the given operator schemata, suitably instantiated and ordered; further, the plan must be applicable in the execution system's current environment.

The inputs to a planning algorithm are a set of operator schemata, an initial state, and a goal. If the input languages for operators, states, and goals are sufficiently expressive, then there is no algorithm which can produce a plan in time polynomial in the size of the inputs. This isn't surprising, since planning problems can encode tasks of general purpose computation (Chapman, 1985). Thus in order to synthesize plans, planners must *search* through a space of alternatives, considering various possible solutions, re-

[‡]This work has been partially supported by the Artificial Intelligence Research Program of the Air Force Office of Scientific Research.

[§]Also affiliated with the Computer Science Department at Rutgers University.

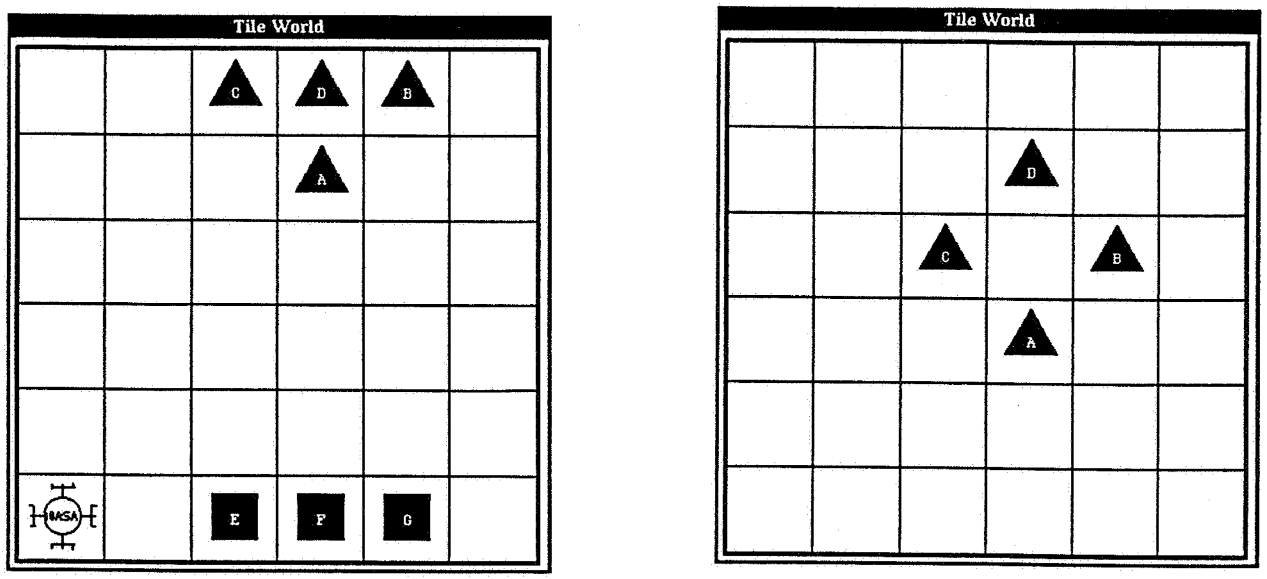


Figure 1: Problem Initial State (left) and Problem Goal State (right)

jecting some, pursuing others. To be useful, a planner must ruthlessly prune its search space: as with any other combinatorially explosive problem, heuristics must be employed to effectively control search. In traditional AI planning, a heuristic which was found to be particularly useful for planning problems was means-ends analysis (Newell & Simon, 1963). This heuristic was sanitized and popularized by the STRIPS system (Fikes & Nilsson, 1971), and versions of means-ends analysis have been used extensively ever since.

During search, most planners represent a plan as a partially ordered set of actions. A partially ordered plan is typically used as shorthand: all total orders consistent with the partial order are considered acceptable. For execution purposes however, plans are often compiled into another form. Abstractly, the form is that of a function which maps a state into the action that should be performed in that state.

For a totally ordered plan, a state-to-action function can be compactly represented as a *triangle table* (Fikes, Hart, & Nilsson, 1972). The triangle table format gives an execution system the ability to index the action most appropriate to execute in its current state. Using a triangle table, an execution system will sometimes repeat failed actions and skip actions in the sequence whose execution is no longer necessary.

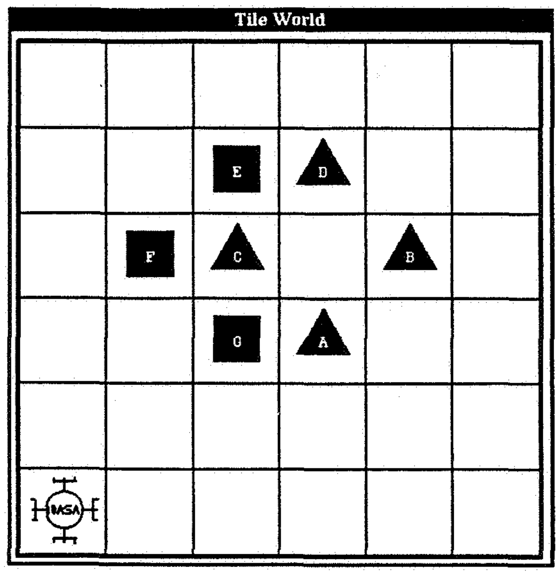
In terms of state coverage, the logical extreme of a triangle table has been called a *universal plan* (Schoppers, 1987). In the abstract, a universal plan is a total function from the space of states to possible actions; that is, the plan that the function characterizes tells the execution system what to do in *all possible* states. A universal plan is an interesting theoretical extreme, and can serve as a holy grail to planners concerned with absolute plan robustness. Of course, for domains of realistic complexity, *knowledge* and *time* will be limiting resources: it will often be impossible to completely define the space of domain states in advance due to lack of knowledge; also, even when the domain can be adequately defined in advance, the computational cost of finding an action for each state can be prohibitively expensive (Ginsberg, 1990).

A Planning and Control Problem

There are several commonly acknowledged limitations of traditional AI planning. This section presents a "simple" planning and control problem for which traditional AI techniques prove inadequate. The reasons why they prove inadequate are addressed in the next section.

Let us consider a domain in which the environment is a two-dimensional grid of cells populated with tiles and a single agent. A tile is a named polygon which fits in a single cell. No matter where the agent is, it can sense the contents of each cell at will. The agent fits in a single cell and has four grippers which extend in the four compass directions. The agent can move horizontally or vertically and can grasp a tile in a horizontally or vertically adjacent cell; the agent can always release a tile that it is grasping.

In addition to the agent-executable actions, there is an external event over which the agent has no control; this event corresponds to a "gust of wind". Winds operate as vectors of force originating from one of the four grid borders. A tile can be "blown" by a gust of wind only if the following two conditions hold: (i) the path



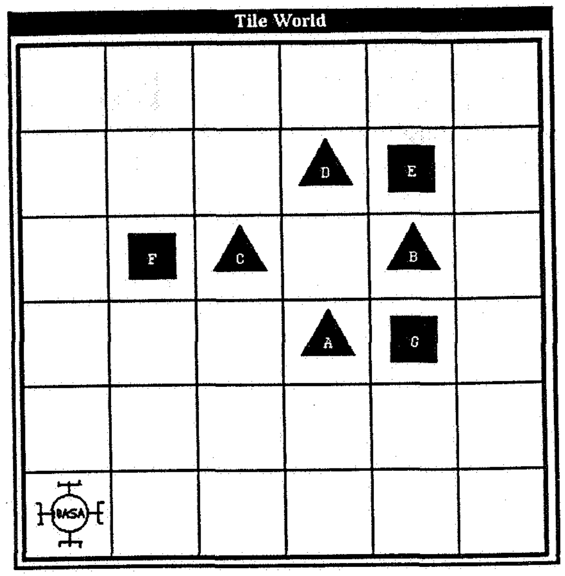


Figure 2: Wind Break Solution (left) and Tile Buttress Solution (right)

between the tile and the wind's origin is clear; and (ii) the cell into which it would be blown is empty. There is a delay specified that applies to all winds, indicating the time lag between successive gusts. Additionally, each of the four directions is assigned a number indicating the probability that a gust of wind will come from that direction.

For example, if the gust delay is 5 minutes and the probability of a south wind is 0.75 and of west wind is 0.25, then every 5 minutes there will be a gust of wind and 75% of those gusts will come from the south and 25% will come from the west.

Let's consider an example problem in this domain which is posed to the agent at 11:52 AM. The current state of the world is depicted in the left half of Figure 1; the agent is in the lower left corner. In this problem, the gust delay is 10 minutes and all gusts come from the west. The goal of the agent is to maintain the configuration of triangular shaped tiles (including the empty inner cell) illustrated in the right half of Figure 1 from noon till midnight. Due to the west wind, simply assembling the goal configuration will not suffice; tiles A, C, and D could be blown from their goal locations between noon and midnight.

One way to prevent the wind from disturbing the goal configuration is to construct a stable wind break using tiles E, F, and G; this type of solution is shown in the left half of Figure 2. The right half of Figure 2 shows an alternate solution using a combination of wind breaks and tile buttresses.

Limitations of Traditional AI Planning

How would traditional AI planning address such a problem? The goal languages of traditional planners do not easily allow for the expression of goals with temporal extent, so it would be difficult to express the requirement that tiles A, B, C, and D remain in their goal locations from noon until midnight. Traditional AI planning has used domains such as the blocks world to study the role of heuristics like means-ends analysis. A typical goal would call for the *achievement* of a certain block assembly. Such goals can be effectively managed by means-ends analysis, since the relationship between an action's effects and the reason for including that action in a plan is relatively straightforward. However, for the problem considered here, where the goal is one of *maintenance*, the relationship between the effects of actions and the satisfaction of the overall goal is less direct. Means-ends analysis requires extensions to be used on goals of maintenance.

It would also be difficult for a traditional planner to represent and reason about the winds that are part of this problem. In traditional AI planning, it is assumed that the only source of change in the world is the execution of a plan step. Traditional planning does not provide reasoning mechanisms for exogenous events; this is especially true for our problem, where the winds occur with statistical regularity. If one is unable to represent the statistical properties of the winds, then one will also be unable to exploit this information to control search. All gusts are not equally likely, so search can focus first on those that have the highest probability of occurring. Traditional planning makes no provision for the use of probabilities in controlling search.

A "closed-loop controller" based on traditional AI planning ideas must generate a complete plan prior to execution. The plan will be passed to the executor for physical realization, and error reports will be fed back to the planner. In response to an error the planner will return another complete plan and begin the cycle anew. This approach, while abstractly adequate, will not work when there are tight deadlines associated with the problem's goals. If the planner has not produced a complete plan by noon, then the executor will have no plan to execute and thus will be unable to take action in service of the goal. When the total time taken to produce and execute a complete plan violates a given deadline, the only workable approach is to allow execution to begin before planning is finished.

ERE: Extending the Traditional Framework

The Entropy Reduction Engine project is our on-going research effort concerned with the effective integration of planning, scheduling, and control. This section briefly examines aspects of the ERE approach which overcome the limitations of traditional AI planning discussed in the previous section.

As an extension to simple goals of achievement, our approach employs a language of *behavioral constraints* which is based on a branching temporal logic (Drummond, 1989). Here is a behavioral constraint, or BC, that expresses the goal in the problem presented above.

```
(maintain
  (and (tile-loc A (3 2))
       (tile-loc B (4 3))
       (tile-loc C (2 3))
       (tile-loc D (3 4))
       (cell-empty (3 3)))
  1200 2400)
```

The predicate `(tile-loc A (3 2))` indicates that the tile denoted by `A` be at the grid location denoted by the pair `(3 2)`. The predicate `(cell-empty (3 3))` is used in the obvious way. The `maintain` part of the BC applies to the conjunction of tile location predicates and the cell empty predicate; this conjunction must be true from the time denoted by `1200` until the time denoted by `2400`.

In order to consider future possible courses of action, our planner needs a *causal theory* for each domain of application. A causal theory is a set of operator schemata which defines both the actions that are controllable by the system and the exogenous events over which the system has no control. In our framework, a causal theory describes the different possible outcomes of an action or event and their associated probabilities. The inclusion of probabilities within our operator language is a significant extension to traditional representations.

One part of the planner, the *projector*, explores various possible futures by repeatedly finding applicable operators and applying them to produce new hypothetical states. The projector creates a directed acyclic graph called a *projection* where each node denotes a domain state and each arc is labeled with a domain operator. Projection associates a duration with each operator application and uses this to calculate a time stamp for the resulting state. The process of creating a projection from a causal theory is called *temporal projection.*

A path in a projection graph denotes a future possible behavior. Projection paths which satisfy a given behavioral constraint are compiled into a set of Situated Control Rules, or SCRs (Drummond, 1989). These are if-then rules, in which the antecedent refers to facts about the execution system's environment and the current behavioral constraint, and in which the consequent contains a set of alternative operators to execute. Available SCRs indicate to the executor those actions which will lead to the eventual satisfaction of its current behavioral constraint. Our execution system always checks to see if any existing SCRs are appropriate to the current state and given behavioral constraint. If so, the SCRs' advice about what to do next is heeded.

Our approach to SCR synthesis requires two phases of temporal projection. The first phase is carried out by an algorithm called *traverse*, and the second phase by an algorithm called *robustify.* Our *traverse* algorithm uses what we call "behavioral constraint strategies" to help incrementally produce executable SCRs.

A *behavioral constraint strategy* (or BC strategy) is a partial order over a set of behavioral constraints. Behavioral constraint strategies for a given behavioral constraint are produced using domain- and problem-specific planning expertise. The process is beyond the scope of this paper; Bresina and Drummond (1990) give more information. The BC strategy constructed for a given BC indicates a set of subproblems for the planner to satisfy and an order in which to satisfy them. For example, consider the following BC strategy.

1st	`(achieve (tile-loc G (2 2)))`
2nd	`(achieve (tile-loc A (3 2)))`
3rd	`(achieve (tile-loc F (1 3)))`
4th	`(achieve (tile-loc C (2 3)))`
5th	`(achieve (tile-loc B (4 3)))`
6th	`(achieve (tile-loc E (2 4)))`
7th	`(achieve (tile-loc D (3 4)))`

This strategy happens to be totally ordered, and could be used by the system to produce the windbreak solution of Figure 2. Following this strategy, the tiles would be assembled row-by-row, bottom-to-top, and each row would be constructed left-to-right.

Traverse synthesizes a single projection path that satisfies a given BC strategy. The algorithm uses the

BC strategy it is given to limit search. The partial order in a BC strategy is "parsed" by *traverse*; *traverse* must find a path which satisfies each BC in the given BC strategy, in an order consistent with the strategy's partial order. When working on the satisfaction of a single BC in a strategy, *traverse* uses a search heuristic based on a combination of path reliability and path utility. The reliability estimate is based on the probabilities of the various state-to-state transitions in the path. Path utility is measured by estimating the remaining work necessary to satisfy a particular BC in the strategy from the current end state of the developing path. In traditional means-ends analysis, a *situation difference* measure was used to estimate the "distance" between an arbitrary state and a goal (Nilsson, 1980). Our notion of estimated remaining work generalizes this simpler notion of situation difference to handle goals with temporal extent (Drummond & Bresina, 1990).

Recall from our motivation in the last section that it might be necessary for an execution system to take action before a complete plan has been generated. Our approach provides this capability by compiling SCRs from each projection subpath that satisfies a BC in the BC strategy. If necessary, the execution system can act on these SCRs. It is the role of the BC strategy to enforce global coherence over the set of local subpaths. Thus, it is the BC strategy which provides some degree of confidence that the developing solution path is likely to be a prefix of a complete solution path.

For example, the wind break BC strategy given above can be used by *traverse* to synthesize a set of SCRs for the placement of tile G in its intended location before the entire set of SCRs for the overall problem has been produced. This means that the execution system can, if necessary, move G into place while further projection is carried out.

Once the entire BC strategy has been satisfied, the execution system will have a set of SCRs describing a single correct behavior. This initial set of control rules has a certain probability of satisfying the given goal. In the second phase of our approach, we use an algorithm called *robustify* to incrementally increase the probability of BC satisfaction by synthesizing additional control rules that handle "error" states the execution system is likely to encounter when following the initial SCRs. Put simply, *robustify* scans along the original satisfactory projection path looking for high-probability "deviations". A deviation from a path is a transition from a state on the path to a state that is not on the path. *Robustify* attempts to recover from such deviations by finding alternative paths back to the original path. Additional SCRs are compiled from each additional recovery path found.

A triangle table (Fikes *et al.*, 1972) is like a set of SCRs designed to deal with each state in a sequence of states, and a universal plan (Schoppers, 1987) is like a set of SCRs which has 100% coverage of the space of states. By using probabilities the algorithm achieves a computationally effective balance between the limited robustness of triangle tables and the absolute robustness of universal plans.

Ours is not the only attempt in AI to address the problem of real-time embedded control. Other representative approaches include Brooks' (1985) *subsumption architecture*, the *action nets* of Nilsson *et al* (1990), Maes' (1990) spreading activation approach, and the *situated automata* of Rosenschein and Kaelbling (Rosenschein, 1989; Rosenschein & Kaelbling 1986; Kaelbling, 1987a,b, 1988). Each of these approaches gives a designer a language and methodology for specifying a control system. Other work in AI has also considered the use of probabilities and temporal reasoning; for instance, see Dean and McDermott (1987), Hanks (1990), and Dean and Kanazawa (1988). Drummond and Bresina (1990) provide a more detailed description of the *traverse* and *robustify* algorithms and comparison with related work.

Conclusions

Our work is attempting to extend the tools of traditional AI planning to handle more complex domains. In particular, we have studied: goals with temporal extent; the representation of, and reasoning mechanisms for, the management of exogenous events; the use of probabilities for controlling search; and the role of plans in guiding an execution system. This research is in early stages, and will benefit from a better understanding of work in the area of discrete event control systems (Ramadge & Wonham, 1989).

Situated Control Rules are used by the execution system as a set of local instructions constituting a partially defined control program. Over time, the precision of the control program increases, and the probability of the execution system "doing the right thing" goes up. This means that, if necessary, the plan execution system can act before a complete plan has been generated. Our approach is predicated on the claim that for many problems there are strict limits on the system's knowledge of the environment and limits on the time given to the system for the computation of a plan. Our approach to managing search has addressed the issue of time-limited computation, but has not presented any solutions to the problem of limited knowledge of the environment.

We are critically concerned with managing the combinatorial complexity of search required to synthesize plans in domains such as the one we have briefly considered in this paper. Algorithms which are polynomial in the number of possible states are too expensive for practical use: heuristics must be used to dramatically cut the number of alternative states considered, and our work on *traverse* and *robustify* represents one attempt to do this.

The problem considered in this paper is drawn from a class of domains we call *the NASA Tile World.* We have implemented a simulator for this class of domains in Common Lisp on Sun workstations.[1] We have also implemented a causal theory for the problem considered in this paper and are now attempting to collect empirical evidence for the utility of our approach to the synthesis of control rules.

Acknowledgements

Andy Philips has contributed significantly to our theory and implementation. Additional members of the ERE group, namely, Smadar Kedar, Rich Levinson, Nancy Sliwa, and Keith Swanson have also provided help along the way. Discussions with Peter Cheeseman and Pat Langley have been informative and entertaining. Thanks to Lisa Dent and John Allen for comments on this paper. Final responsibility for all errors and omissions rests, of course, with the authors.

References

[1] Bresina, J., and Drummond, M. 1990. Integrating Planning and Reaction: A Preliminary Report. *Proceedings of the 1990 AAAI Spring Symposium Series* (session on Planning in Uncertain, Unpredictable, or Changing Environments).

[2] Brooks, R. 1985. A Robust Layered Control System for a Mobile Robot. Technical Report 864, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts.

[3] Chapman, D. Nonlinear Planning: a Rigorous Reconstruction. In *Proceedings of IJCAI-85*, pp. 1022-1024, Los Angeles, CA, 1985. International Joint Committee on Artificial Intelligence.

[4] Dean, T., and Kanazawa, K. 1989. A Model for Projection and Action. *Proceedings of IJCAI-89*. pp. 985–990.

[5] Dean, T., and McDermott, D. 1987. Temporal Database Management. *AI Journal.* Vol. 32(1). pp. 1–55.

[6] Drummond, M., and Bresina, J. 1990. Anytime Synthetic Projection: Maximizing the Probability of Goal Satisfaction. *Proceedings of AAAI-90.* Boston, MA.

[7] Drummond, M. 1989. Situated Control Rules. *Proceedings of Conference on Principles of Knowledge Representation & Reasoning.* Toronto, Canada.

[8] Ginsberg, M. 1989. Universal Planning: An (Almost) Universally Bad Idea. *AI Magazine*, Vol. 10, No. 4. pp. 40–44.

[9] Hanks, S. 1990. Projecting Plans for Uncertain Worlds. Yale University, CS Department, YALE/CSD/RR#756.

[10] Fikes, R., Hart, P., and Nilsson, N. 1972. Learning and Executing Generalized Robot Plans. *AI Journal*, Vol 3, pp. 251–288.

[11] Fikes, R. and Nilsson. N. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *AI Journal*, Vol. 2, pp. 189–208.

[12] Kaelbling, L. 1987a. An Architecture for Intelligent Reactive Systems. ***Reasoning About Actions and Plans.*** M. Georgeff and A. Lansky, Eds., Morgan Kauffman.

[13] Kaelbling, L. 1987b. REX: A Symbolic Language for the Design and Parallel Implementation of Embedded Systems. *Proceedings of AIAA Conference on Computers in Aerospace.* Wakefield, Massachusetts.

[14] Kaelbling, L. 1988. Goals as Parallel Program Specifications. *Proceedings of the Seventh National Conference on Artificial Intelligence.* St. Paul, Minnesota.

[15] Maes, P. 1990. How To Do the Right Thing. *Connection Science Journal.* (Special Issue on Hybrid Systems. J. Hendler, editor).

[16] Newell, A. and Simon, H.A. 1963. GPS: a Program that Simulates Human Thought. In Feigenbaum, E.A. and Feldman, J. (eds) *Computers and Thought* McGraw-Hill, New York.

[17] Nilsson, N., Moore, R., and Torrance, M., ACTNET: An Action Network Language and its Interpreter. Draft paper, Stanford Computer Science Department, February 1990.

[18] Nilsson, N. 1980. *Principles of Artificial Intelligence.* Tioga Publishing Company, CA.

[19] Ramadge, P. and Wonham, W. 1989. The Control of Discrete Event Systems. *Proceedings of the IEEE.* Vol. 77, No. 1 (January). pp. 81–98.

[20] Rosenschein, S. 1989. Synthesizing Information-Tracking Automata from Environment Descriptions. *Proceedings of Conference on Principles of Knowledge Representation & Reasoning.* Toronto, Canada.

[21] Rosenschein, S. and Kaelbling, L. 1986. The Synthesis of Digital Machines with Provable Epistemic Properties. *Proceedings of Workshop on Theoretical Aspects of Knowledge.* Monterey, CA (March 13-14).

[22] Schoppers, M. 1987. Universal Plans for Reactive Robots in Unpredictable Environments. *Proceedings of the Tenth International Conference on Artificial Intelligence.* pp. 1039–1046, Milan, Italy.

[1]The simulator is available free by anonymous FTP; contact the authors for information.